Übungsblatt 4 - More with Graphics App

Zuletzt bearbeitet von Jürgen Hahn

Übungsblatt 4 - More with Graphics App

Wichtige Informationen zur Bearbeitung der Aufgabe

- Informationen zur Entwicklungsumgebung *IntelliJ IDEA*
- Informationen zum Im- und Export von Projekten
- GraphicsApp

Starterpaket

Ein vorbereitetes Starterpaket zur selbständigen Implementierung der Aufgabe finden Sie hier.

Refactoring durch Methoden

Klasse im Starterpaket: `Target`

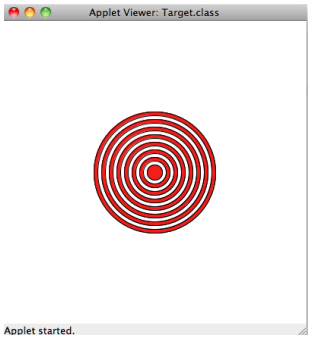

Abbildung 1: Target

Im Starterprojekt für dieses Übungsblatt finden Sie eine Musterlösung für das Target-Programm aus der letzten Übung. Bauen Sie den Code so um, dass er leichter verständlich, modularer und besser wartbarer wird:

1. Ergänzen Sie die Methode `getRingColor`, die eine Ganzzahl erwartet und eine Ringfarbe zurückgibt. Die zurückgegebene Ringfarbe ist davon abhängig, ob die übergebene Zahl gerade oder ungerade ist.
2. Ergänzen Sie die Methode `drawRing`, die den Ring mit der übergebenen Farbe und dem übergebenen Radius zeichnet.

Passen Sie die `for`-Schleife innerhalb der `drawDartTarget`-Methode an, indem Sie den vorhandenen Code durch die neu implementierten Methoden ersetzen. Testen Sie, ob Ihr neu strukturiertes Programm immer noch korrekt läuft.

Ball mit Physik

Klasse im Starterpaket: `BouncingBall`

Erstellen Sie eine `GraphicsApp`, die einen Ball von links nach rechts über die Zeichenfläche springen lässt:

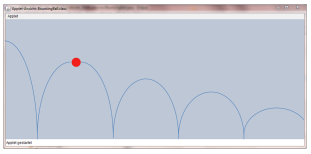

Abbildung 2: Bouncing Ball

Der Ball startet in der linken oberen Ecke, fällt dann zu Boden und prallt wieder ab, solange er nicht den Bildschirm verlassen hat.

Auf den Ball wirken zwei Einflussfaktoren:

- Die Schwerkraft *drückt* von oben auf den Ball und erhöht dessen Geschwindigkeit beim Herunterfallen in jedem Animationsschritt um einen konstanten Wert (z.B. 0.15). Beim Aufsteigen wird die Geschwindgkeit des Balles analog reduziert, bis der Ball bei der Geschwindgkeit '0' den Scheitelpunkt erreicht hat.
- Bei jedem Aufprall gibt der Ball einen Teil seiner Energie ab. Legen Sie eine Konstante an, die angibt, wieviel Energie (= vertikale Geschwindigkeit) der Ball nach dem Aufprall auf den Boden noch behält (z.B. 0.9).

Die horizontale Geschwindigkeit des Balls ist während des gesamten Programms konstant. Speichern Sie den Ball und seine Geschwindigkeit (d.h. Positionsveränderung bei jedem Animationsschritt) in geeigneten Instanzvariablen ab. Bei jedem Durchlauf von draw() sollte Ihr Programm den Ball auf seine neue Position setzen und anschließend überprüfen, ob der Ball auf den Boden geprallt ist. Wenn der Ball auf den Boden getroffen ist, dann verändern Sie seine Geschwindigkeit.

Achtung: Es kann sein, dass der Ball ein gutes Stück unterhalb des Bodens gesprungen ist, wenn Sie eine Kollision bemerken. Wenn dies der Fall ist, müssen Sie ihn nach der Kollision wieder über den Boden hieven. Dies können Sie mit dem folgenden Code erreichen:

```
double diff = ball.getY() - (getHeight() - ball.getHeight());
ball.move(0, -2 * diff);
```

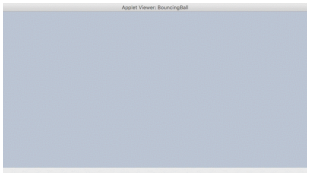

Abbildung 3: Bouncing Ball in Motion

Farbige Zufallskreise

Entwickeln Sie eine `GraphicsApp`, die 100 farbige Kreise auf der Zeichenfläche darstellt. Jeder der Kreise hat eine zufällige Farbe, einen zufälligen Durchmesser zwischen 5 und 50 Pixeln und eine zufällige Position auf der Zeichenfläche. Die Kreise müssen alle innerhalb der Zeichenfläche dargestellt werden. Verwenden Sie die in der Vorlesung vorgestellte Klasse `Random` um alle Zufallswerte zu erzeugen.

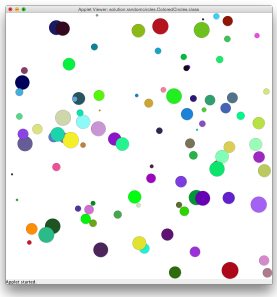

Abbildung 4: Random Circles

Implementieren Sie eine eigene Methode pro zufälliger Eigenschaft der Kreise und speichern Sie den Rückgabewert in einer lokalen Variable, bevor Sie den Konstruktor der Klasse Ellipse aufrufen:

- `private Color getNextColor()`
- `private Point getNextPoint()`
- `private int getNextDiameter()`

Verwenden Sie (wie immer) geeignete Konstanten, um unveränderliche Werte abzuspeichern.

Circle Rain, Circle Rain

In dieser Aufgabe geht es darum, mit zwei Arrays die Animation von einer beliebigen Anzahl an Kreisen zu verwalten. Über eine Konstanten soll festgelegt werden, wieviele Kreise gezeichnet werden. Entsprechend sollen weitere Konstanten bestimmen, wie breit ein Kreis ist, so dass alle nebeneinander auf die Zeichenfläche passen.

Alle Kreise haben eine Startposition am oberen Bildschirmrand. Zu Beginn des Programms sollen jeweils ein Array mit den Kreisen und ein Array mit Geschwindigkeiten angelegt werden. Beide Arrays sollen die gleiche Größe haben. Die Geschwindigkeiten und die Farbe der Kreise sind zufällig zu wählen.

In der `draw()`-Schleife sollen nun alle Kreise mit der ihnen entsprechenden Geschwindigkeit vertikal nach unten bewegt werden. Stößt ein Kreis am unteren Rand an, so ist seine Position so anzupassen, dass er wieder am oberen Bildschirmrand startet.

In der folgenden GIF-Animation ist das Ergebnis zu sehen:

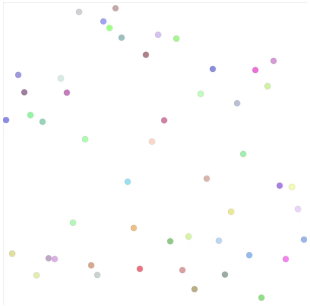

Abbildung 5: Circle Rain